Problem 1) Python Programming

Write a function called `print_every_other_word` that takes in a filename and prints out every other word that appears on each line starting with the first word.

For example, if we had a file named poem.txt with the contents:

> *she sells*
> *sea shells by*
> *the sea shore*

Then if we were to call `print_every_other_word('poem.txt')`, it would print:

> *she*
> *sea by*
> *the shore*

Problem 2) Tabular Data

For the following problems, we will use a Pokemon dataset similar to the one we used in the second homework assignment. As a reminder, the dataset looks something like this:

id	name	level	type	weakness	atk	def
59	Charizard	55	fire	water	110	66
121	Starmie	67	water	electric	174	60
125	Electrabuzz	37	electric	ground	64	62
110	Weezing	30	poison	psychic	168	29
23	Ekans	81	poison	psychic	30	108

2.a.i) `max_atk_difference_manual`

First, imagine that the table above is stored as a list of dictionaries in a variable called `data`. Write a function `max_atk_difference_manual` that takes a list of dictionaries as a parameter and returns the largest difference in the attack between any two Pokemon in the table.

For example, if we called `max_atk_difference_manual(data)`, it would return 144. Your solution should run in O(n) time where n is the number of rows in the table. If there are fewer than two pokemon in the dataset, this function should return None.

2.a.ii) `max_atk_difference_pandas`

Write a function called `max_atk_difference_pandas` that behaves exactly like 2.a.i, except that it takes a pandas `DataFrame` of the dataset as a parameter. You may not use any loops in your solution.

2.b) `best_matchup_manual`

Hunter isn't very good at Pokemon, and he needs some help finding the best matchup between two pokemon. In order to win more Pokemon battles, for a given Pokemon p, Hunter needs to figure out which Pokemon has the lowest attack for each type of Pokemon that p is not weak against.

Write a function called `best_matchup_manual` which takes a list of dictionaries and a row number as a parameter and returns a dictionary mapping each pokemon type, except the one that Hunter's pokemon is weakest against, to the name of the pokemon with the lowest attack in that type.

For example, assuming we have parsed the data above in a variable called `data`, if we were to call `best_matchup_manual(data, 0)` it would return:

```
{'electric': 'Electrabuzz', 'Poison': 'Ekans'}
```

Notice that Starmie is not included in the returned dictionary because it is a water type - the kind that Hunter's pokemon is weakest against. Weezing is not included because Ekans is the poison pokemon with the lowest attack.

Problem 3) Machine Learning

How can you tell the difference between puppers, doggos, and woofers? This classic question has plagued humanity for years -- maybe a decision tree can help us figure it out!!

Imagine we have the following dataset, in `dogs.csv`. We want to use the average weight, fur length, and energy level to predict the type.

	breed	average weight	fur length (1-5)	energy level (1-5)	type
0	basset hound	50	1	1	doggo
1	bernese mountain dog	90	4	3	woofer
2	french bulldog	25	1	3	pupper
3	great dane	150	1	4	woofer
4	mastiff	180	2	3	woofer
5	pembroke corgi	25	3	5	doggo
6	pug	15	1	2	pupper
7	standard schnauzer	40	5	3	doggo
8	shiba inu	20	3	5	doggo

3.a) Train a classifier

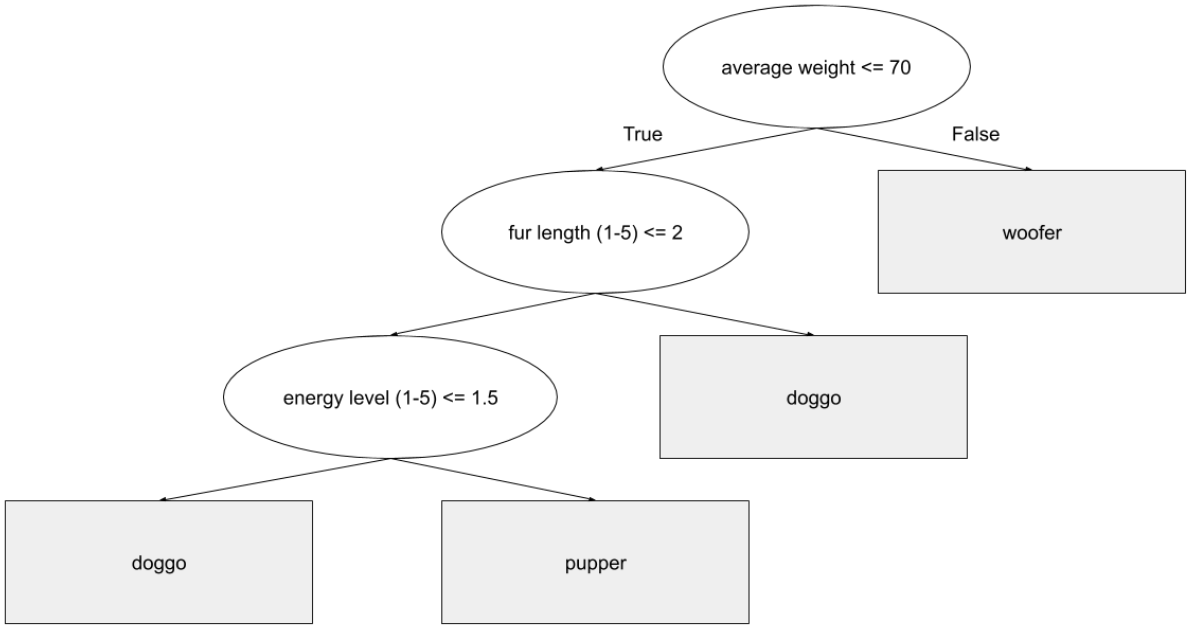
Write the code below to read in the CSV file as a pandas `DataFrame`, split the data into features `X` and labels `y`, and create a `DecisionTreeClassifier` trained on your `X` and `y`. This does not need to be written in a function - just write the code directly. For this problem, don't worry about splitting into train and test sets since we won't be doing any programmatic prediction. Assume that we have already imported `DecisionTreeClassifier` from `sklearn.tree`.

3.b) Featuring

Why might we choose to not include the breed feature in the `X` used to train our decision tree? Explain your answer in one or two sentences.

3.c) Making Predictions

On decision tree that can be learned on the features described in 3.a is shown below



What class would predict for the following examples?

- `breed: 'Siberian Husky', average weight: 55, fur length: 4, energy level: 5`

- `breed: 'Boston Terrier', average weight: 16, fur length: 1, energy level: 3`

Problem 4) Classes

For this problem, you will be implementing a class called `Rectangle` that represents a geometric rectangle with floats representing the height and width.

4.a) Implementing `Rectangle`

Implement the following methods with the described arguments. You should not include any additional arguments for these methods.

Method	Description
`__init__(self, width, length)`	Given two numbers, width and length, constructs a `Rectangle` with that width and length.
`area(self)`	Returns the area of this `Rectangle`. Recall that area is defined by width times height.
`scale(self, scale_factor)`	Multiplies the height and width of this `Rectangle` by a number `scale_factor.` This means if we originally had a 2x4 `Rectangle`, and scaled it by a factor of 3, it would then be a 6x12.
`__eq__(self, other)`	Returns `True` if the other represents the same `Rectangle` (has the same height and width). Returns `False` otherwise. You may assume that other is always a `Rectangle`

4.b) Using `Rectangle`

Write a short program that does the following:

- constructs a `Rectangle` with width 2 and height 3.
- Scales it by a factor of 2
- Prints the area
- constructs a second `Rectangle` with width 4 and height 6.
- Prints "Same" if the first `Rectangle` is the same as the second `Rectangle`, and prints "Different" otherwise.

You do not need to write a main method for this problem.

Problem 5) Big-O Efficiency

For the following problems, write the run-time of each function using the Big-O notation. For these problems, we will use n as the variable to describe the length of the input structure. Your answer should be the “smallest” Big-O runtime possible (i.e. you may not say $O(n^{12})$ as an answer if O(n) is an answer that is closer to the actual run-time).

5.a)

```
def method1(n):
  sum = 0
  for i in range(0, n // 2)
    sum += i
  return sum
```

5.b)

```
def method2(n):
  result = 0
  while n // 2 > 0:
    print(n)
    result *= 5 - n // 2
    n = n // 2
  return result
```

5.c)

```
def method3(n):
  val = 32
  if n < 100:
    for i in range(n):
      val += i * 3
  return val * 3
```

5.d)

```
def method4(n):
  lst = [i * 3 for i in range(n)]
  val = 0
  for element in lst:
    val += element
    for i in range(n):
      lst[i] *= i
  return val, min(lst)
```